УДК 519.5

©2010. **Е. А. Татаринов**

Базовый алгоритм восстановления конечного графа

Рассматривается задача восстановления графа агентом, перемещающимся по его ребрам, считывающим и изменяющим метки на элементах графа. Предложен базовый метод восстановления. Алгоритм требует 2 различные краски и кубического, от числа вершин графа, числа шагов. Найдены модификации алгоритма, которые понижают верхнюю оценку временной сложности. Найдены операции над графами, результирующий граф которых имеет верхнюю оценку сложности выполнения базового алгоритма не хуже, чем исходный.

Ключевые слова: *восстановление графов, агент, метка вершины, блуждание по графу.*

1. Введение. Рассматривается задача распознавания конечного, неориентированного графа, без петель и кратных ребер при помощи агента, который перемещается по нему и изменяет метки на вершинах и ребрах графа [1]. Данная работа посвящена анализу базового алгоритма, предложенного в [4]. Показывается, что алгоритмы, предложенные в [5–7], являются модификациями алгоритма, предложенного в [4]. Рассматриваются классы графов, на которых понижается верхняя оценка сложности выполнения базового алгоритма или его модификаций. Найдены операции над графами, результирующий граф которых имеет порядок верхней оценки сложности выполнения не хуже, чем исходный.

2. Необходимые определения и понятия. Рассматриваются конечные, неориентированные графы, без петель и кратных ребер. Все неопределяемые понятия общеизвестны и их можно найти, например, в [2 - 4]. Пусть $G(V, E)$ - граф, у которого V - множество вершин, E - множество ребер, $E \subseteq V \times V$, мощности n и m, соответственно. Ясно, что $m \leq \frac{n(n-1)}{2}$. Тройку $((u, v), v)$ будем называть инцидентором ("точкой прикосновения") ребра (u, v) и вершины v. Множество таких троек обозначим I. Множество $L = V \cup E \cup I$ назовем множеством элементов графа G. Краской будем называть ресурс, который имеется у агента в неограниченном количестве, а камнем - ресурс, который имеется у агента в единичном экземпляре. Краски и камни образуют множество меток - M, которые могут быть использованы при раскраске элементов графа G. Если элемент графа метится некоторой краской, то предыдущая краска стирается и вместо нее наносится новая. Если элемент графа метят камнем, то существующая на нем краска не уничтожается, а становится "невидимой" до тех пор, пока камень будет находиться на элементе. Функцией раскраски графа G назовем отображение $\mu : L \to M$, где $M = \{w, b, r, 1.., p\}$, w интерпретируется как белый цвет (краска), r- красный, b- черный, а множество чисел $1, .., p$ интерпретируется как множество камней.

Последовательность $u_1, u_2, .., u_k$ попарно смежных вершин называется путем в графе G , а k - длина пути. При $u_1 = u_k$ этот путь называется циклом. Окрестностью $O(v)$ вершины v будем называть множество элементов графа, состоящее

из вершины v, всех вершин u смежных с v, всех ребер (u, v) и всех инцидентов $((v, u), v), ((v, u), u)$. χ – цикломатическое число графа равное $m - n + 1$. Изоморфизмом графа G и графа H назовем такую биекцию $\phi : V_G \to V_H$, что $(u, v) \in E_G$ точно тогда, когда $(\phi(v), \phi(u)) \in E_H$. Изоморфные графы равны с точностью до обозначения вершин и раскраски их элементов.

Мобильный агент A (далее просто A) характеризуется следующими свойствами. Он передвигается по графу из вершины u в вершину v по ребру (v, u). При этом он может изменить окраску вершин u, v, ребра (v, u), инцидентов $((v, u), v)$, $((v, u), u)$ метками из множества M. Находясь в вершине v A воспринимает ("видит") метки всех элементов окрестности $O(v)$ и на этом основании определяет по какому ребру (v, u) он будет перемещаться и как будет перекрашивать элементы из $O(v)$. A обладает конечной, неограниченно растущей внутренней памятью, в которой фиксируется результат его функционирования на каждом шаге, и, кроме того, выстраивается представление графа G, вначале неизвестного A, списками ребер и вершин. Вершину, в которой A находится в данный момент, будем называть текущей. Будем говорить, что A "зациклился" , если он попал в вершину, из которой он может попасть только в ранее посещенные вершины. Откатом будем называть отход A по вершинам красного пути (далее r - путь) из его конца к его началу или же из некоторой вершины r - пути к его концу при восстановлении обратных ребер. Длина отката зависит от длины r - пути, длина которого не более, чем n.

Нумерацией $F : V(G) \to N$ вершин графа G называется инъективное отображение множества его вершин во множество N натуральных чисел. Номер вершины $u \in V_G$ в нумерации F обозначается $F(v)$. M - нумерацией вершин графа называется нумерация вершин графа, соответствующая порядку их обхода при поиске в глубину [4]. Древесными ребрами называются [4] ребра, порождающие дерево поиска при обходе графа в глубину, остальные ребра - обратные. Вершиной сочленения будем называть вершину v такую, что после ее удаления и инцидентных ей ребер граф становится несвязным [9].

$T(n)$ назовем временной сложностью выполнения алгоритма, а $T^*(n)$ и $T_*(n)$, соответственно, верхней и нижней оценкой временной сложности. $S(n)$ - емкостная сложность выполнения алгоритма.

3. Постановка задачи. Пусть задан класс K графов конечных, неориентированных, без петель и кратных ребер, все элементы которых окрашены краской w. Требуется разработать такой алгоритм функционирования A (т.е. передвижения по графу и раскраски его элементов), что он, будучи помещен в произвольную вершину произвольного неизвестного ему графа $G \in K$, через конечное число шагов построит граф H, изоморфный G, т.е. восстановит G.

4. Алгоритм восстановления графа.

4.1. Стратегия. Предлагаемый алгоритм основан на стратегии поиска в глубину. Предлагаемая стратегия обладает рядом особенностей, по сравнению с известными [2 - 4]. Во - первых, граф G A не известен и A на каждом шаге обладает информацией о красках элементов из окрестности $O(v)$ текущей вершины v. Во - вторых, при прохождении вершин графа G A создает неявную M - нумерацию

пройденных вершин: при первом посещении вершины u она окрашивается красным цветом и ей фактически ставится в соответствие следующий номер по порядку. На основе этой нумерации и происходит построение графа H изоморфного G, с точностью до меток на элементах. В процессе обхода графа G A строит неявное дерево поиска в глубину и относительно этого дерева все ребра разделяются на: древесные и обратные. Древесные ребра проходятся, по крайне мере, 2 раза: при первом прохождении A метит один из инцидентов ребра красной краской, а при последнем проходе он окрашивает оба инцидентора ребра черным цветом. Обратные - проходятся один раз и при первом прохождении оба инцидентора ребра метятся черной краской.

Древесные ребра восстанавливаются при первом проходе A по ним. Красные вершины графа G, на каждом шаге алгоритма, образуют простой r - путь длины не более, чем n. С помощью этого пути распознаются обратные ребра, при проходе в новую вершину r - путь удлиняется, при проходе назад - укорачивается, при распознавании обратного ребра - не изменяется. Вершина, у которой все инцидентные ребра восстановлены, красится черным. Алгоритм завершает работу, когда r - путь становиться пустым, а все вершины черными.

4.2. "Базовый алгоритм". В описании алгоритма v - текущая вершина графа G, V_H, E_H - списки вершин и ребер графа H, - счетчик числа посещенных вершин графа G. $k(1), k(2), .., k(t)$ - список номеров вершин r - пути, t - длина этого списка, j - счетчик, используемый для восстановления обратных ребер.

1. Сч:= 1;
2. $t := 1$
3. $k(t)$:=Сч;
4. $V_H := \{1\}$;
5. $E_H := \varnothing$;
6. A красит: $\mu(v) := r$;
7. *if* в $O(v)$ есть ребро (v, u) с $\mu((u, v), v) = w$ *then goto* 8 *else goto* 9;
8. *if* в $O(v)$ есть ребро (v, u), у которого $\mu(v, u) = w$ и $\mu(u) = \mu(v) = r$ *then* ВОССТ(v) *else* ВПЕРЕД(v)
9. *if* в $O(v)$ есть ребро (v, u) с $\mu(((v, u), v)) = r$ *then do* НАЗАД(v); *goto* 7; *end do*;
10. A красит: $\mu(v) := b$;

ВПЕРЕД(v)

1. A выбирает из $O(v)$ произвольное ребро (v, u), у которого $\mu(u) = \mu(v, u) = w$, и переходит по нему в вершину u;
2. $\mu(v, u) := r$, $\mu(u) = r$, $\mu((v, u), u) := r$;
3. Сч:=Сч+1;
4. $k(t + 1)$:=Сч;
5. $t := t + 1$;
6. $v := u$;
7. $V_H := V_H \cup \{$Сч$\}$;
8. $E_H := E_H \cup \{k(t - 1), k(t)\}$;

НАЗАД(v)

1. A выбирает ребро $(v,u) \in O(v)$, у которого $\mu(v,u) = r$, и $\mu((v,u),v) = r$, переходит по нему в вершину u.

2. A красит: $\mu(v) := b, \mu(v,u) := b$;

3. $v := u$;

4. $t := t - 1$;

5. Из списка $k(1), .., k(t+1)$ удаляется элемент $k(t+1)$;

ВОССТ(v)

1. A выбирает из $O(v)$ произвольное ребро (v,u), у которого $\mu(v) = \mu(u) = r$ и $\mu(v,u) = w$, и переходит по нему в вершину 146 u;

2. A красит: $\mu(v,u) := b$;

3. A выбирает из $O(v)$ ребро (u,z), у которого $\mu(u,z) = r$ и $\mu((u,z),z) = r$, и переходит по нему в вершину z;

4. $u := z$;

5. $j := 1$;

6. *While* в окрестности $O(u)$ есть ребро (u,z), у которого $\mu(u,z) = r$ и $\mu((u,z),z) = r$ *do*

7. A переходит по ребру (u,z) в вершину z;

8. $u := z$;

9. $j := j + 1$; *end do*

10. $E_H := E_H \cup \{(k(t), k(t-j))\}$;

Подробно базовый алгоритм описан в [5].

Теорема 1. *Выполняя "Базовый алгоритм" , A восстановит любой граф $G \in K$ с точностью до изоморфизма, за время не большее, чем $O(n^3)$. При этом используется 2 краски и не более, чем $O(n^2)$ ячеек памяти. Нижняя оценка временной и емкостной сложности равны $O(n)$.*

Доказательство теоремы приведено в [5].

При выполнении процедуры ВПЕРЕД(v) и НАЗАД(v) A проходит одно ребро, при выполнении процедура ВОССТ(v) A проходит одно обратное ребро и не более $n-1$ ребер r - пути, т. е. цикл, состоящий из обратного ребра и некоторого конечного отрезка r - пути, соединяющего вершины, инцидентные обратному ребру. При выполнении алгоритма A проходит дерево поиска в глубину, состоящее из $n-1$ ребер, и $m-n+1$ обратных ребер. Будем считать, что инициализация алгоритма (строки 1 - 6 описания алгоритма восстановления), анализ окрестности $O(v)$ текущей вершины и выбор одной из трех процедур (строки 7 - 9) занимают некоторое постоянное число единиц времени. Так же будет считать, что выбор ребра при выполнении процедур и проход по ребру осуществляется за 1 единицу времени. Тогда временная сложность алгоритма определяется суммой следующих величин:

1. Инициализация выполняется один раз и ее сложность равна $O(1)$;

2. Процедура ВПЕРЕД(v) выполняется ровно $n-1$ раз и общее время ее выполнения оценивается как $O(n)$;

3. Общее время выполнения процедуры НАЗАД(v) оценивается как $O(n)$;

4. Выполнение процедуры ВОССТ(v) один раз оценивается как $O(n)$, а ее выполнение для всех обратных ребер как $O(n(m-n+1))$, т.е. как $O(n^3)$.

Значит, $T^*(n)$ есть величина $O(n^3)$.

Емкостная сложность $S(n)$ алгоритма определяется сложностью списков V_H, E_H, $k(1),..,k(t)$, сложность которых, соответственно, определяется величинами $O(n), O(n^2)$, $O(n)$. Следовательно: $S(n) = O(n^2)$.

5. Модификации "Базового алгоритма"."Базовый алгоритм" имеет достаточно высокий порядок верхней оценки временной сложности и, поскольку число вершин графов неограниченно, A требуется конечная, но бесконечно растущая память. Модификации "Базового алгоритма" дают лучшие оценки временной и емкостной сложностей.

В настоящей работе рассматриваются следующие модификации "Базового алгоритма":1) разделить A на два агента: агента исследователя $АИ$, с конечной памятью, и агента экспериментатора $АЭ$; 2) восстанавливать обратные ребра, сразу все для одной вершины; 3) восстанавливать обратные ребра, при "зацикливании" A; 4) Разделить множества ребер на непересекающиеся классы,которые восстанавливаются различными стратегиями.

Подробно эти методы описаны в [5 - 7]. Целью настоящего раздела - показать, что "Базовый алгоритм" является основой этих модификаций.

5.1. Модификация 1. Поскольку при обходе графа и разметки его элементов A не пользуется записанной ранее в память информацией, а только оперирует с ней на определенных этапах обхода и разметки, возможно разделение A на два агента $АИ$ и $АЭ$. Первый будет перемещаться по графу G, считывать и изменять метки на элементах графа, и передавать сообщения $АЭ$ о своих передвижениях и изменениях меток. Второй будет принимать сообщения, обрабатывать их и на их основании строить граф H. Принципиально "Базовый алгоритм" не изменяется, изменяется, только то, что $АИ$ будет передавать соответствующее сообщение и не будет строить в своей памяти представление графа H. Различных видов сообщений, которые передает $АИ$, будет ограниченное количество, поэтому $АИ$ может быть реализован конечным автоматом. Данная модификация позволяет $АЭ$ строить граф H либо в процессе приема сообщения от $АИ$, либо после того, как $АИ$ отправит последнее сообщение. Алгоритм, где применяется данная модификация подробнее описан в [6]. Данная модификация не меняет порядка $T^*(n)$, а только ослабляет ограничения накладываемые, на $АИ$.

5.2. Модификация 2. При стратегии восстановления всех обратных ребер, инцидентных вершине u, за одно выполнение процедуры ВОССТ(v) изменится только процедура ВОССТ(v). В ней A будет метить исследуемую вершину u камнем l (признак того, что для данной вершины восстанавливаются обратные ребра), а затем отходить от нее по r - пути к его началу, для поиска белых ребер, смежных с вершиной u, у которой $\mu(u) = l$. При этом подсчитывается количество переходов до каждого момента, когда A видит вершину $\mu(u) = l$. Затем на основании неявной M - нумерации происходит восстановление всех обратных ребер вершины u и по r - пути $АИ$ возвращается в его конец. При этом процедура ВОССТ(v) вызыва-

ется только один раз для каждой вершины графа, вне зависимости от количества обратных ребер, смежных с ней. Тогда вызов процедуры BOCCT(v) будет производиться не $m-n+1$ раз (количество обратных ребер), а n раз (для каждой вершины один раз). В этом случае $T^*(n)$ будет величиной $O(n^2)$, однако A потребуется один дополнительный камень l.

5.3. Модификация 3. Обратные ребра можно восстанавливать как при первом их появлении, а так же при "зацикливании" A. При этом не произойдет изменения ни $T^*(n)$, ни $S(n)$ и A потребоваться то же самое количество камней, однако это повлияет на то, в какой момент времени будет вызываться процедура BOCCT(v) , т.е. обратные ребра будут восстановлены в графе H.

5.4. Модификация 4. Множество вершин графа G разобьем на два множества: ординарные и неординарные вершины. Вершина $v \in V_G$ ординарная, если $deg(v) \leq D$, где D - заранее известная константа. В противном случае вершина неординарная. Для того, чтобы A на графе G мог разбить множество вершин на два непересекающихся класса, необходимо выполнения условия: для всех $(u, v) \in E_G$, либо v, либо u – ординарная вершина. В этом случае множество ребер разбивается на три непересекающиеся множества: древесные, обратные и неординарные, т.е. те, которые смежны с неординарной вершиной. Обратные и древесные ребра для ординарных вершин восстанавливаются, так же как и в "Базовом методе" или его модификации 1 - 3. Обратных ребер у ординарной вершины не более, чем D, A может их подсчитать и при выполнении процедуры BOCCT(v) подсчитывать количество восстановленных ребер и прекратить отход по вершинам r - пути, когда он восстановит все обратные ребра вершины, не доходя до начала r - пути. Неординарные ребра восстанавливаются так: неординарное ребро, по которому A попал в неординарную вершину и пометил ее камнем i из множества камней $\{1, .., p\}$, восстанавливается при первом прохождении. В момент пометки вершины камнем i, в графе H с камнем i ассоциируется следующая по порядку вершина, и, таким образом, ей присваивается неявный номер. Все остальные неординарные ребра неординарной вершины, помеченной i камнем, восстанавливаются, когда A попадает в вершину r - пути, из которой видно неординарную вершину, помеченную камнем i, в этом случае для неординарного ребра неявные номера смежных с ним вершин вычисляются без дополнительных переходов. Алгоритм, реализующий данную модификацию, подробно описан в [6]. Поскольку A строит представление графа в виде списка ребер графа , то в случае, если p - заранее известная константа, количество ребер будет величиной $O(n)$ и тогда $S(n)$ будет величиной $O(n)$, что на порядок лучше, чем в общем случае.

6. Частные случаи классов графов и операции над ними. В данном разделе будут рассмотрены некоторые классы графов и операции над ними, такие, что при выполнении на них "Базового алгоритма" или его модификаций 1 - 4 возможно будет понизить верхнюю оценку временной или емкостной сложности, либо она не будет ухудшена.

6.1. Классы графов. Рассмотрим следующие классы графов.

Пусть K - класс графов конечных, неориентированных, без петель и кратных ре-

бер, элементы которых можно метить специальными красками. Это самый широкий класс графов, все остальные классы являются его подклассами.

Пусть ${K^P}_D$ - класс графов, содержащих p неординарных вершин, а степень ординарных не превышает D. Отдельного внимания заслуживает случай, когда $p = 0$, обозначим его K_D.

Лемма 1. *Верхняя оценка емкостной сложности восстановления графа $G \in K_D$ "Базовым алгоритмом" или его модификациями 1 - 4 является величиной $O(n)$.*

Доказательство. A строит граф H в виде списка ребер. Поскольку $m \leq Dn$, то для хранения потребуется $2m$ ячеек памяти и не более, чем $n-1$ ячеек для хранения списка вершин r - пути. Таким образом, $S(n)$ будет величиной $O(n)$. □

Замечание 1.Утверждение леммы справедливо и для графа $G \in {K^p}_D$, при условии, что p заранее известная константа. Поскольку количество неординарных ребер не превосходит pn, то $S(n)$ будет величиной порядка $O(n + pn) = O(n)$.

Лемма 2. *Верхняя оценка временной сложности "Базового алгоритма" на графе $G \in K_D$ является величиной $O(n^2)$.*

Доказательство. Поскольку граф $G \in K_D$, то в нем будет не более, чем Dn обратных ребер, для каждого из которых будет выполнять процедуру ВОССТ(v) , которая выполняется не более, чем за n шагов. Тогда $T^*(n)$ будет величиной $O(n^2)$. □

6.2. Операции над графами. Рассмотрим операции над графами, при применении которых верхняя оценка результирующего графа не ухудшается.

6.2.1. Операция отождествления. Отождествление двух вершин v_1 и v_2 графов G_1 и G_2 из класса K, порождают граф G , в котором вершины v_1 и v_2 объединяются в вершину сочленения u. Частный случай отождествления, когда один из графов G_1 или G_2 состоит из двух вершин, соединенных ребром. Такое отождествление фактически является добавлением к графу висячей вершины. Отождествление будет правильным, если: 1) не будут отождествляться ординарная и неординарная вершины; 2) не будет нарушаться ординарность получаемой вершины сочленения; 3) нарушение ординарности увеличит D на константу. Далее будем рассматривать только правильные отождествления.

Лемма 3. *Граф G, полученный отождествлением двух графов G_1 и G_2, восстанавливается за время не большее, чем $T_1^*(n) + T_2^*(n)$, где $T_1^*(n)$ и $T_2^*(n)$ – верхние оценки временной сложности восстановления G_1 и G_2, соответственно.*

Доказательство. Рассмотрим все возможные варианты начала выполнения агентом A алгоритма восстановления G. Пусть u_1 вершина сочленения G_1 и G_2. Существует три различных начальных положения A : 1) A попал в вершину u_1; 2) A попал в вершину $v_1 \in V(G_1)\backslash\{u_1\}$; 3) A попал в вершину $v_2 \in V(G_2)\backslash\{u_1\}$. Рассмотрим эти случаи.

1. A добавляет вершину u_1 в r - путь. После этого он либо попадет в вершину $v_1\prime \in V(G_1)\backslash\{u_1\}$, либо в вершину $v_2\prime \in V(G_2)\backslash\{u_1\}$. Предположим, что он попал в $v_1\prime$ ($v_2\prime$). Поскольку вершина u_1 добавлена в r - путь, то A может попасть из вершин,

содержащихся в $V(G_1)\backslash\{u_1\}$ ($V(G_2)\backslash\{u_1\}$) в вершины, содержащиеся в $V(G_2)\backslash\{u_1\}$ ($V(G_1)\backslash\{u_1\}$), только после того, как он посетит их все, восстановит инцидентные им ребра и вернется в u_1. A восстановит подграф G, который совпадает с графом G_1 (G_2). Затем он вернется в u_1 и перейдет в вершину $v_2\prime$ ($v_1\prime$) и продолжит обход графа. Таким образом, A восстановит подграф G, который совпадает с графом G_2 (G_1). Тогда $T^*(n) = T_1^*(n) + T_2^*(n)$.

2. A добавит вершину v_1 в r - путь. В этом случае существует два различных варианта дальнейшего выполнения алгоритма.

2.1. A из v_1 начинает выполнение алгоритма и выполняет его для подграфа G, который совпадает с G_1. Затем он попадает в вершину u_1, а из нее в вершину $v_2\prime \in V(G_2)\backslash\{u_1\}$ и продолжает выполнять алгоритм, для подграфа G, который совпадает с G_1. Следовательно, $T^*(n) = T_1^*(n) + T_2^*(n)$.

2.2. A из v_1 начинает восстановление и выполняет его для подграфа G, который совпадает с $G_1\prime$, подграфом G_1. Затем он попадает в вершину u_1, а из нее в вершину $v_2\prime \in V(G_2)\backslash\{u_1\}$ и продолжает восстанавливать подграф G, который совпадает с G_2. После перехода в вершину u_1 A добавит ее в r - путь, и из подграфа G, который совпадает с G_2, A уйдет только после того, как полностью его восстановит. После чего A вернется в вершину u_1 и продолжит восстановление графа, для оставшейся части $G = G_1\backslash G_1\prime$. В силу выше сказанного $T^*(n) = T_1^*(n) + T_2^*(n)$. Следовательно, в этом случае утверждение теоремы для G выполняется.

3. Этот случай полностью аналогичен случаю 2. Воспользуемся рассуждениями из пункта 2., рассматривая вместо графа G_2 граф G_1, а вместо графа G_1 граф G_2.

Из пунктов 1 -3 следует справедливость леммы. □

Лемма 4. *Граф G, полученный добавлением в граф G_1, из некоторого класса, висячей вершины имеет порядок сложности выполнения "Базового алгоритма" и его модификаций 1 - 4, такой же, что и G_1.*

Доказательство. В добавленную висячую вершину можно будет попасть только по добавленному вместе с ней ребру. Значит, добавленное ребро может быть только древесным и для него A выполнит 2 шага. Таким образом, $T^*(n)$ изменится на величину $O(2)$, что не изменит порядок ее сложности. □

Замечание 2. Если в граф последовательно добавить k висячих вершин, то $T^*(n)$ изменится на величину $O(2k)$, что не изменит порядок ее сложности.

6.2.2. Введение вершины "Штейнера". Введение в граф "вершины Штейнера" [8] - добавление вершины между двумя соединенными ребром вершинами с последующим удалением соединяющего ребра и соединением ребрами добавленной вершины и исходных.

Лемма 5. *Граф G, полученный добавлением в граф G_1, из некоторого класса,вершины "Штейнера" имеет порядок сложности выполнения "Базового алгоритма" и его модификаций 1 - 4, такой же, что и G_1.*

Доказательство. Предположим, что между вершинами v_1 и v_2 была добавлена вершина u, ребро (v_1, v_2) было удалено из графа G_1, а ребра - (v_1, u) и (v_2, u) добавлены.

Рассмотрим, как изменится $T^*(n)$, в зависимости от того, было ли ребро (v_1, v_2) древесным или обратным при выполнении A восстановления графа G_1.

1. Если ребро (v_1, v_2) было древесным, то (v_1, u) и (v_2, u) в графе G так же будут древесными, поскольку A в вершину u может попасть из вершины v_1, а выйти из нее, только в вершину v_2, и наоборот. Значит, A потребуется выполнить еще дополнительно 4 шага, в этом случае к $T^*(n)$ прибавится величина $O(4)$, что не изменит порядка ее сложности.

2. Если ребро (v_1, v_2) было обратным, то рассмотрим, появятся ли в графе дополнительные циклы, которые приведут к ухудшению $T^*(n)$. Поскольку при выполнении этой операции удаляется одно ребро и добавляется два новых, то $m = m_1 + 1$ и $n = n_1 + 1$, тогда $\chi = m - n + 1 = m_1 + 1 - (n_1 + 1) + 1 = m_1 - n_1 + 1 = \chi_1$. Значит, дополнительных циклов не появится, и верхняя оценка временной сложности не ухудшится. □

6.2.3. Классы графов, порожденные операциями над графами. Рассмотрим классы графов, порожденные специфическим отождествлением нескольких графов из некоторого класса.

Пусть T^k - класс квази - деревьев. Будем говорить, что граф $G \in T^k$, если он образован последовательным отождествлением графов $G_j \in K$, $j = 0, .., k$ где $k \in N$, и $u_0, .., u_{k-1}$ - вершины сочленения, если, отождествив G_j, $j = 0, .., k$ с вершинами, а вершины $u_0, .., u_{k-1}$ с ребрами, мы получим граф вида дерево.

Лемма 6. *Граф G, полученный последовательным отождествлением графов $G_j \in K$, $j = 0, .., k$, восстанавливается за время не большее, чем $\sum_{j=1}^{k} T_j{}^*(n)$, где $T_j^*(n)$ – верхняя оценки временной сложности восстановления G_j.*

Доказательство. Для графа, полученного последовательным сочленением $G_j \in K$, $j = 0, .., k$ где $k \in N$, и $u_0, .., u_{k-1}$ - полученные вершины сочленения, можно провести такие же рассуждения, на этапе построения, как и для G_1 и G_2 из леммы 4, для полученного после j - ого сочленения графа и следующего для сочленения графа G_{j+1}. Значит, утверждение верно. □

Пусть S^k - класс квази - колец. Будем говорить, что граф $G \in S^k$, где $k \in N$, если он образован сочленением G_{j-1} с G_j, $j = 1, .., k$, и G_0 с G_k, где $G_j \in K$, $j = 0, .., k$, $u_0, .., u_k$ - вершины сочленения, если, отождествив G_j, $j = 0, .., k$ с вершинами, а вершины $u_0, .., u_k$ с ребрами, мы получим граф вида - цепочка вершин соединенных по кругу.

Лемма 7.*Пусть $G \in K$. Разобьем множество его вершин на два не пересекающихся подмножества $V_2(G)$ и $V_3(G)$, где множество $V_2(G) = \{v \in V(G) : deg(v) \leq 2\}$, а множество $V_3(G) = \{v \in V(G) : deg(v) \geq 3\}$, $V(G) = V_2(G) \cup V_3(G)$. Тогда $T^*(n)$ есть величина $O(\sum_{v \in V_3(G)} deg(v)n)$.*

Доказательство. Для доказательства рассмотрим два подмножества множества вершин графа G, и выясним, в каких случаях они порождают откаты при восстановлении обратных ребер.

Рассмотрим множество вершин $V_3(G)$. Каждая вершина из $V_3(G)$ не может породить откатов более, чем ее степень. Действительно, откаты возникают при восста-

новдении обратных ребер вершины. При этом, если $v \in V_3(G)$ вызывает "зацикливание то она порождает откат. Если эта вершина из множества $V_3(G)$ не вызывает "зацикливания" то она будет смежна с вершиной, которая вызовет "зацикливание" и откат. Значит, каждая вершина $v \in V_3(G)$ может вызвать откат не более, чем $deg(v)$ раз. Тогда будет не более, чем $|V_3(G)|deg(v)$ откатов.

Рассмотрим множество вершин $V_2(G)$. Они порождают откаты в двух случаях.

1. Вершина из $V_2(G)$ является вершиной, из которой A начинает выполнение "Базового алгоритма". В этом случае, A восстановит только ребра из ее окрестности. Это ребро, по которому A совершил проход в глубину графа. И одно не пройденное ребро, по которому он может попасть в начальную вершину. Значит, вершина, которой оно инцидентно, вызовет "зацикливание" и откат. Таким образом, вершина из $V_2(G)$, будучи начальной, может вызвать откат. Так как эта вершина может содержаться либо на последовательности вершин, которые соединяют две вершины из множества $V_3(G)$, в этом случае откат, вызываемый этой вершиной, уже учтен при подсчете откатов, порождаемых вершинами из множества $V_3(G)$, либо на цепочке вершин, которая примыкает к вершине из множества $V_3(G)$, в этом случае откат, порождаемый вершиной из множества $V_2(G)$, учтен при подсчете откатов для вершин из множества $V_3(G)$. Значит, вершины из множества $V_2(G)$ могут породить не более одиного отката и не повлияют на $T^*(n)$.

2. Вершина из $V_2(G)$ не является вершиной начальной для выполнения алгоритма, но смежна с вершиной из множества $V_3(G)$. Откаты, порождаемые этой вершиной, были учтены при подсчете количества откатов для ребер вершин из множества $V_3(G)$. Таким образом, согласно известной нам верхней оценке временной сложности "Базового алгоритма" $2n - 1 + (m - n + 1)n$. Слагаемое $(m - n + 1)n$ соответствует откатам. Тогда сложность можно переписать с учетом количества откатов, а их $\sum_{v \in V_3(G)} deg(v)$ штук. Следовательно, $T(n) \leq 2n - 1 + \sum_{v \in V_3(G)} deg(v)n$, а при достаточно больших n, $T^*(n)$ есть величина $O(\sum_{v \in V(G)_3} deg(v)n)$. □

Лемма 8. *Пусть $G \in S^k$, тогда $T^*(n)$ есть величина $\sum_{j=1}^{k} T_j^*(n)+$ $O(\sum_{v \in G_0} deg(v)n)$, где $G_j \in K$ $j = 0, .., k$, где $T_j^*(n)$– верхняя оценка временной сложности выполнения "Базового алгоритма" на графе G_j, множество $V_3(G_j)$ определено как и в лемме 8, а G_0 подграф, соответствующий части графа, в которую был помещен A изначально.*

Доказательство. Для данного класса графов существенным является начальное положение A . Поэтому, не нарушая общности рассуждений, предположим, что A попадает в вершину $v \in V(G_0)\backslash\{u_0\}$. Если же A попал в вершину $v \in G_j$, j$\neq$ 0, перенумеруем G_j и $u_0, .., u_k$ так, чтобы $j = 0$. Легко видеть, что при разъединении графа G по вершине u_k мы получим граф $G\prime \in T^k$, а если удалить из G граф, соответствующий графу G_0, то мы получим граф $G\prime\prime \in T^{k-1}$.

Рассмотрим случаи: A изначально попадает в вершину u_k, A попадает в вершину $v \in V(G_0)\backslash\{u_0, u_k\}$.

1. A изначально попал в вершину u_k. Этим A фактически разъединит граф G по вершине u_k. Рассмотрим граф $G\prime$, полученный в результате этого. Для этого

графа справедлива лемма 6 и рассуждения, приведенные при ее доказательстве. Таким образом, убрав из рассмотрения вершину u_k, можно утверждать, что $T^*(n)$ есть величина $\sum_{j=1}^{k} T_j^*(n)$. Однако, необходимо учесть действия, которые выполняются с вершиной сочленения. Поскольку она добавлена в r - путь, то действия по восстановлению ребер $e_1 = (u_k, v)$, где $v \in V(G_0)$, могут происходить только в рамках подграфа, соответствующего G_0 и уже учтены в слагаемом $T_0^*(n)$. Для ребер $e_2 = (u_k, u)$, где $u \in V(G_k)$, вершина u_k будет вызывать "зацикливание"и откаты как в пределах подграфа, соответствующего G_k, так и длины, соизмеримой с n, поскольку для того, чтобы пройти в u_k по ребрам, инциденторы которых помечены r, A придется пройти по вершинам, принадлежащим всем G_j, где $j = 0, .., k$. Первые откаты учтены в слагаемом $T_k^*(n)$. Количество же вторых не превысит $deg(u_k)$. Значит, их можно учесть, добавив к сумме величину $O(deg(u_k)n)$. Тогда, $T^*(n)$ есть величина $\sum_{j=1}^{k} T_j^*(n) + O(deg(u_k)n)$.

2. A изначально попал в $v \in V(G_0)\backslash\{u_0, u_k\}$. В этом случае A выполнит следующие действия: восстановит часть или весь подграф, соответствующий G_0, после чего попадет в вершину u_0 (u_k), а из нее пройдет далее в вершину $u \in V(G_1)$ ($u \in V(G_k)$), распознает весь или часть подграфа, соответствующего G_1 (G_2), и так далее, до тех пор, пока он не попадет в вершину u_k (u_0), добавив ее в r - путь, и пойдет далее в вершину $u \in V(G_k)$ ($u \in V(G_0)$). Таким образом, A фактически разъединит граф G, убрав из него подграф, соответствующий G_0. Рассмотрим граф G'', полученный в результате этого. Для него будет справедлива лемма 7. Вершины подграфа, соответствующего G_0, могут вызывать "зацикливание" и откаты при восстановлении обратных ребер, как в пределах подграфа G_0, так и длины, соизмеримой с n. Поскольку при "зацикливании" в 1 - окрестности вершины могут находиться вершины, в которые можно пройти по ребрам, инциденторы которых помечены r, предварительно пройдя по всем вершинам подграфов G_j, где $j = 0, .., k$. Таким образом, $T^*(n)$ будет состоять из $T^*(n)$ для G'' и слагаемого, в котором будут учтены откаты для вершин подграфа, соответствующего G_0. Их будет не более $\sum_{v \in V(G_0)} deg(v)$ штук, длины не более n. Таким образом, $T^*(n)$ есть величина $\sum_{j=1}^{k} T_j^*(n) + O(\sum_{v \in V_3(G_0)} deg(v)n)$.

Исходя из 1 – 2 можно заключить, что T^* есть величина $\sum_{j=1}^{k} T_j^*(n) +$ $O(\sum_{v \in V_3(G_0)} deg(v)n)$, поскольку слагаемые в пункте 1. $T_0^*(G)$ и $O(deg(u_k)n)$ поглощаются слагаемым $O(\sum_{v \in V(G_0)} deg(v)n)$ из пункта 2. □

В [6] подробно рассматривались дополнительные частные случаи графов и операции над ними для модификации 4 "Базового алгоритма" , из - за громоздкости описания и доказательства в данном разделе они рассматриваться не будут.

7. Выводы. Предложен базовый метод восстановления ранее неизвестного графа с точностью до меток на его элементах графов при помощи передвигающемуся по нему агента. Данный метод использует две краски и имеет временную сложность $O(n) \leq T(n) \leq O(n^3)$. Базовый метод допускает модификации, которые позволяют получить лучшую оценку временной сложности и упрощают агента A, однако, накладывают дополнительные ограничения на восстанавливаемый граф. Найдены операции над графами такие, что результирующий граф имеет верхнюю оценку временной сложности не хуже, чем исходные.

1. *Dudek G., Jenkin M.* Computational principles of mobile robotic // Cambridge Univ. Press. –2000 – 280 p.
2. *Ахо А., Хопкрофт Дж., Ульман Дж.* Построение и анализ вычислительных алгоритмов. – М.: Мир, 1979. – 536 с.
3. *Кормен Т., Лейзерсон Ч., Ривест Р.* Алгоритмы: построение и анализ. – М.: МЦНМО, 2001. – 960 с.
4. *Касьянов В. Н., Евстигнеев В. А.* Графы в программировании: обработка, визуализация и применение. – СПБ.: БХВ – Петербург, 2003. – 1104 с.
5. *Грунский И. С. , Татаринов Е. А.* Распознавание конечного графа блуждающим по нему агентом. // Вестник Донецкого университета. Серия А. Естественные науки –, 2009, Вып. 1. – С.492-497.
6. *Грунский И. С., Татаринов Е. А.* Алгоритм распознавания графов. // Тр. 4 междунар. конф. "Параллельные вычисления и задачи управления" PACO'2008, Москва, Ин-т проблем управления им. В. А. Трапезникова РАН, – 2008 – С.1483-1498.
7. *Татаринов Е. А.* М - нумерация, как метод распознавания графов. //Збірник наукових праць "Питання прикладної математики математичного моделювання" , Дніпропетровськ – 2010. – С.260-272.
8. *Кристофидес Н.* Теория графов – алгоритмический подход. – М.: Мир, 1978. – 432 с.
9. *Э. Рейнгольд, Ю.Нивергельт, Н.Део* Комбинаторные алгоритмы. Теория и практика. – М.: Мир, 1980. – 477 с.

E. A. Tatarinov

Basic algorithm for reconstructing a finite graph.

The problem of reconstructing a graph agent moving through his edges, read and modify marks on the elements of the graph. We propose a basic method of reconstruction. The algorithm requires 2 different colors and cube of the number of vertices number of steps. Found modification of the algorithm, which lowers the upper bound of time complexity. Found the operation on graphs, the resulting graph which has an upper bound for the complexity of the basic algorithm is not worse than the original.

Keywords: *graph reconstraction, agent, nod mark, move on graph .*

Є. О. Татаринов

Базовий алгоритм відновлення скінченного графа.

Розглядається задача відновлення графа агентом, який переміщується по його ребрах, що прочитує і змінює мітки на елементах графа. Запропоновано базовий метод відновлення. Алгоритм потребує 2 різні фарби і кубічного, від числа вершин графа, числа кроків. Знайдено модифікації алгоритму, які знижують верхню оцінку часової складності. Знайдено операції над графами, результуючий граф яких має верхню оцінку складності виконання базового алгоритму не гірше, ніж вихідний.

Ключові слова: *відновлення графів, агент, мітка вершини, блукання по графу.*

Ин-т прикл. математики и механики НАН Украины, Донецк
MDgerelo@yandex.ru

Получено 23.11.10